PRACTICAL NO. 4

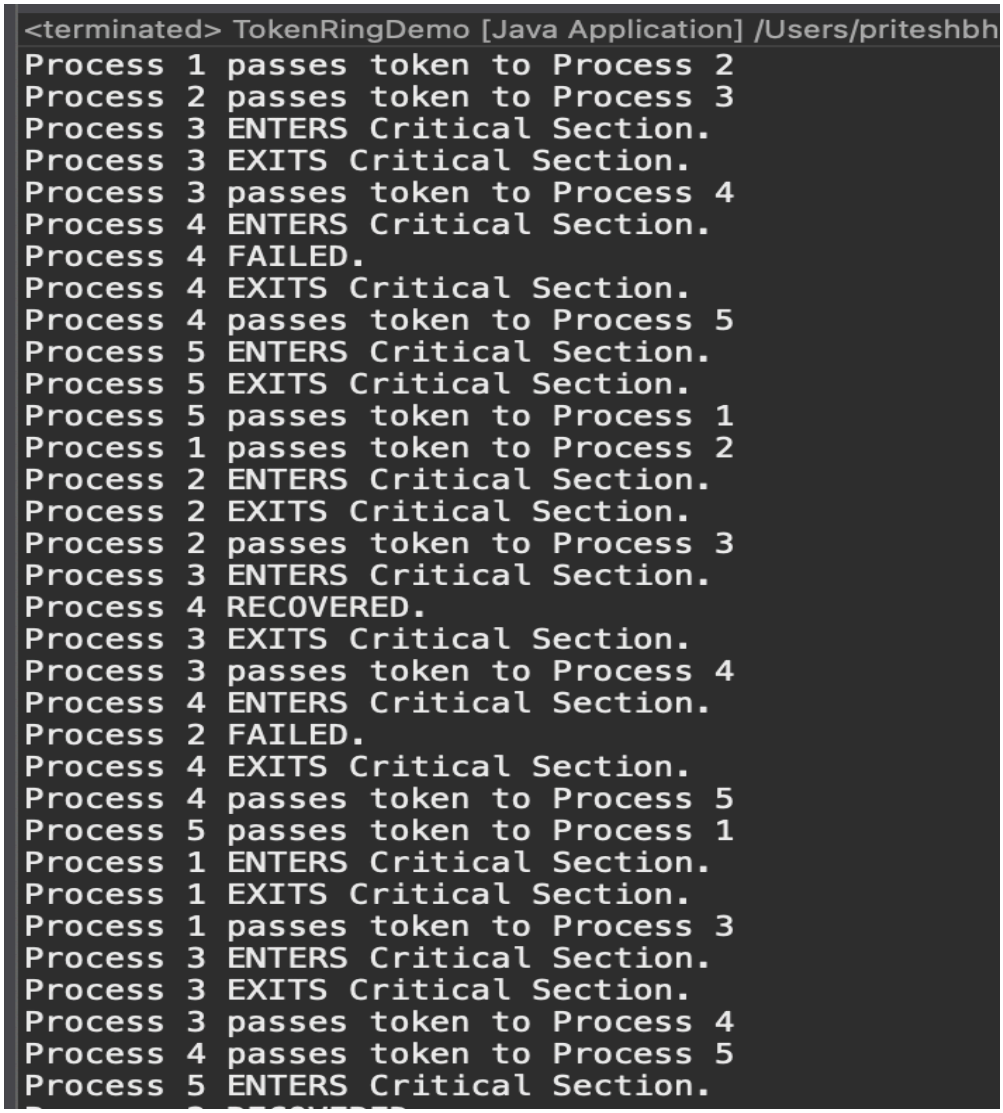
Election & Mutual Exclusion

a) Write a java program to implement mutual exclusion using Token ring algorithm.

```
package ME3;

import java.util.Random;

class Process extends Thread {
    private final int id;
    private volatile boolean hasToken;
    private boolean wantsToEnterCS;
    private Process nextProcess;
    private boolean isAlive = true;

    private static final Random random = new Random();

    public Process(int id) {
        this.id = id;
        this.hasToken = false;
        this.wantsToEnterCS = false;
    }
    public void setNextProcess(Process next) {
        this.nextProcess = next;
    }
    public void receiveToken() {
        if (isAlive) {
            hasToken = true;
        }
    }
    private void enterCriticalSection() {
        System.out.println("Process " + id + " ENTERS Critical Section.");
        try {
            Thread.sleep(3000); // Simulate work inside critical section
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Process " + id + " EXITS Critical Section.");
    }
    public void failProcess() {
        isAlive = false;
        hasToken = false;
        System.out.println("Process " + id + " FAILED.");
    }
    public void recoverProcess() {
        isAlive = true;
        System.out.println("Process " + id + " RECOVERED.");
    }

    @Override
    public void run() {
        while (true) {
            if (hasToken && isAlive) {
                wantsToEnterCS = random.nextBoolean();

                // Enter CS if process wants to
                if (wantsToEnterCS) {
                    enterCriticalSection();
                }
                // Pass the token to the next alive process
                hasToken = false;
```

```
            Process next = nextProcess;
            while (!next.isAlive) {
                next = next.nextProcess; // Skip dead processes
            }
            System.out.println("Process " + id + " passes token to Process " + next.id);
            next.receiveToken();
            // Sleep to simulate delay
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
  }
}

public class TokenRingDemo {
    public static void main(String[] args) {
        int n = 5; // Number of processes
        Process[] processes = new Process[n];

        // Create processes
        for (int i = 0; i < n; i++) {
            processes[i] = new Process(i + 1);
        }

        // Set up the ring
        for (int i = 0; i < n; i++) {
            processes[i].setNextProcess(processes[(i + 1) % n]);
        }

        // Start all processes
        for (int i = 0; i < n; i++) {
            processes[i].start();
        }

        // Give initial token to process 1
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        processes[0].receiveToken();

        // Simulate failures and recoveries
        try {
            Thread.sleep(5000);
            processes[3].failProcess(); // Process 4 fails

            Thread.sleep(8000);
            processes[3].recoverProcess(); // Process 4 recovers

            Thread.sleep(5000);
            processes[1].failProcess(); // Process 2 fails

            Thread.sleep(7000);
            processes[1].recoverProcess(); // Process 2 recovers
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output:



b) Write a java program to implement Election Algorithm.

```
package Election;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Process{
	int id;
	boolean active;
	Process(int id) {
		this.id=id;
		this.active=true;
	}
```

```
}
public class RingElection {
	static List<Process> ringList = new ArrayList<>();
	static int coordinator = -1;

	static void startElection(int startId) {
	  System.out.println("\nProcess " + startId + " starts elction...");
	  List<Integer> activeIds = new ArrayList<>();
	  int n = ringList.size();
	  int current = startId;

	  do {
		Process p = ringList.get(current);
		if (p.active) {
			activeIds.add(p.id);
			System.out.println("Election message at processs "+p.id);
		}
		current =(current +1)%n;
	} while (current!=startId);

	 //highest ID
	  int newCoordinator = Collections.max(activeIds);
	  coordinator=newCoordinator;
	  System.out.println("--> New Coordinator is Process "+ coordinator);
	  current = startId;
	  do {
		Process p = ringList.get(current);
		if (p.active) {
			System.out.println("Coordinator message at process "+p.id);

		}
		current = (current +1)%n;

	} while (current!=startId);
}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		for(int i=1;i<=5;i++) {
			ringList.add(new Process(i));
		}
		coordinator = 5;
		System.out.println("Initial Coordinator: Process "+coordinator);
		ringList.get(4).active = false;  // Changed from 4 to 5
		System.out.println("\nProcess 5 (Coordinator) has Crashed...");
```

```
                startElection(1);
                ringList.get(4).active = true;   // Changed from 4 to 5
                System.out.println("\nProcess 5 has recovered.");
                System.out.println("Process 5 inquires: Current cordinator is Process
"+coordinator);
        }
}
```

Output:

```
Initial Coordinator: Process 5

Process 5 (Coordinator) has Crashed...

Process 1 starts elction...
Election message at processs 2
Election message at processs 3
Election message at processs 4
Election message at processs 1
--> New Coordinator is Process 4
Coordinator message at process 2
Coordinator message at process 3
Coordinator message at process 4
Coordinator message at process 1

Process 5 has recovered.
Process 5 inquires: Current cordinator is Process 4
```